Security Fundamentals

UNIX SysAdmin DeCal Fall 2020
mdcha (adapted w/ <3 from abizer)

Let's Speedrun CS 161 and Then Some.

- For more theoretically theoretical information, take CS 161, it's a fun survey class.
- This lecture doesn't really care about why we choose a certain curve for ECC–we care about the fact that the curve is secure (hopefully).

Why do we care?

- Basic government functions at risk: Voting security sucks
- Hospitals suffering from ransomware attacks (NYT)
- Financial security
  - Equifax breach
- Personal information
- Job security!
  - According to Forbes, computer security market size in 2020 is ~$173 billion, expected to reach $270 billion in 2026.
  - You can lose a 2016 presidential election because your security team improperly classified a spear-phishing email as benign!

What is Security?

Security is keeping systems functioning as intended in the face of adversaries.

This can take multiple forms:

1. Confidentiality
2. Integrity/Authenticity
3. Availability

Confidentiality

- Only those authorized to read a file can do so
- Not any random Joe should have access to your users' data
- Messaging securely (god help you if you send private keys and passwords over Slack)

Integrity/Authenticity

- How do we know the message we received was actually the message the sender intended for us to receive?
- How do we know who we're actually talking to?
- "Hi, this is [user], I accidentally lost my phone and need a new SIM card. Can you send one over to this address?"
    - (lol Twitter)

Availability

- What do you do when some script kiddie gets mad and buys a botnet to DDOS your site?
- A system that is unavailable is as bad as no system at all.
- The internet makes this problem a million times worse.

Basic Principles

- **Security is Economics**
  - No system is 100% secure
  - As much as the OCF would enjoy installing an iris scanner to our server room, it's too damn expensive
- **Least Privilege**
  - Don't give more access than absolutely necessary
  - You guys don't have root access to our servers
    - (tbh neither should I)

Basic Principles

- **Defense in depth**
  - Multiple redundant protections provide layers of security.
  - Certain OCF resources can only be accessed from within our network and then our credentials.
- **Complete mediation**
  - Every form of access must be checked/controlled (We only have one way in here with OpStaff checking IDs)

Basic Principles: Account for Human Factors

- If users don't like it, they won't use it
- Forcing 16+ char passwords to be rotated every two months ends up with them writing it on a sticky note.
- There is oftentimes a tradeoff between security and convenience.
- Historically security tools and procedures are woefully unusable
  - openssl is the worst command you will ever use
  - Traditional public key distribution is awful
    - Have you ever tried using a PGP public key server
    - Keysigning parties
  - some idiots on Twitter telling people to wrap their car keys in tin foil and put it in the freezer....

Threat Modeling

- What are you protecting?
- Who needs access to it?
- How valuable is it?
- Who can attempt to access it? How?
- How much will it cost to protect it?
- What is an attack? Disgruntled users? Natural disasters?

Security Basics

1. Authentication
2. Encryption
3. Hashing
4. Signatures and Certificates

Building Blocks

1. **Authentication:** Am I *really* a DeCal facilitator?
2. **Encryption:** Keep the NSA from reading your files until they get the FBI and their monkey wrench cryptographers involved
3. **Hashing:** Turn Big Data™ into small data
4. **Signatures and Certificates:** Making sure you are who you say you are

Everything is interrelated and builds off of one another

Don’t #$%^& with primitives

- Always use standard libraries and protocols
- Always use standard libraries and protocols
- Always use standard libraries and protocols
- Always use standard libraries and protocols
- Always use standard libraries and protocols
- Always use standard libraries and protocols
- Always use standard libraries and protocols
- Always use standard libraries and protocols

Don’t #$%^& with primitives

- I will personally make sure you fail this DeCal if I catch you trying to implement your own primitives on critical services.
- Some attacks only show themselves after you get a PhD.
  - “All elliptic curve questions are either trivial or research-level” -My number theory professor
- Other attacks only show up after you’ve sold your soul and signed an NDA with the NSA

Authentication

Problem: You have data and you only want to give access to the owner of the data. (e.g. banking info)

How do they identify themselves and prove they are the owner?

Passwords suck.

- People share passwords (disclaimer: I'm also a Netflix moocher)
- Hard to properly store and easy to steal
- Limited search space
  - 8 alphanumeric characters is (36^8) = ~3e12 possibilities, ~1 month to crack.
- "random" is a joke
- Too many damn passwords to memorize
- Most password requirements are stupid as hell

What do we do as users?

- Don’t re-use passwords
  - Do you really trust that shitty website to never leak your password (hi linkedin)
- Use a password manager
  - Do you really want to memorize hundreds of random characters
  - Make your passwords long
  - I use pass because I hate myself

What do we do as users?

- Make your master pass*phrase* long: “TheOCFisNotTheBerkeleyPrintingService”
  - Easy to remember
  - Safe against brute force attacks
    - 52^37 ~ 3e63: 10.07 thousand trillion trillion trillion centuries at 100 trillion guesses / second
    - My alphabet space is tiny!

What do we do as SysAdmins?

- Do everything on the past two slides
- NEVER store plaintext passwords
  - When, not if, you fail
- Store them securely (more on that later)
- Don’t have stupid password rules
  - Follow NIST 800-63B §5.1.1.2B guidelines
- Enforce lockouts or delays on multiple login failures
- Audit your code to make sure no leaks occur

Better Authentication (Multi-Factor Authentication)

- **Something you know**
  a. e.g. PIN, password, mother's maiden name
- **Something you have**
  a. E.g. TOTP, hardware token, U2F, etc.
    i. Google Authenticator app
    ii. YubiKey
  b. Don't use SMS 2FA for the love of god (hi reddit)
- **Something you are**
  a. usually means biometrics, e.g. fingerprint, Apple FaceID, iris scanning, cardiac rhythm, even gait analysis

Managing Authentication and Authorization

Many office workers have to deal with authenticating against multiple services: payroll, mail, code repositories, etc.

How do you manage this?

- Have each user manage their own credentials?
  - What if one person loses their credentials? One hundred?
  - What if someone quits?
  - What if someone uses ‘password’ for all their passwords?
  - How do you manage who gets access to what?

Managing Authentication and Authorization

A small note on definitions:

Authentication : Verifying the identity of the user.
Authorization : Verifying that the user is supposed to access the material.

Not inputting the proper password against CalNet Auth is an authentication issue.
Trying to change my test grades in BCourses as a student is an authorization issue.

Managing Authentication and Authorization: SAML

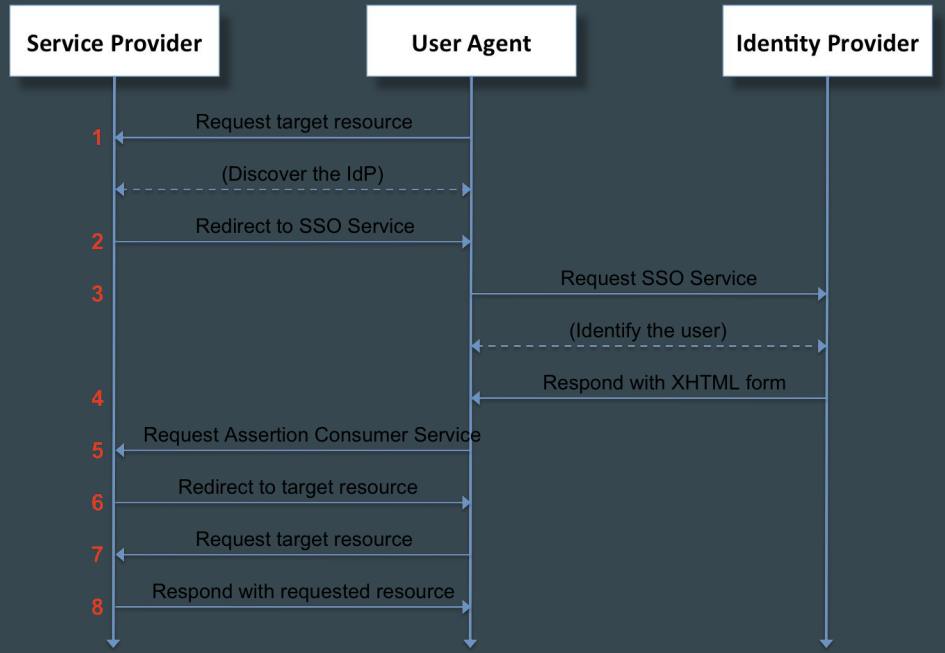
- Service Provider: Mail, Payroll, etc
- User Agent: You/Your web browser
- Identity Provider: Makes sure you're you

Benefits:

- User only needs to remember one login credential!
- The IdP can now more easily manage access to all services
- The IdP can ensure all users follow certain security policies

Downsides:

- The Service Provider needs to actively take part
- XML lol



(Image: Wikimedia/Tom Scavo)

Managing Authentication and Authorization: Kerberos

- Authentication system for services over an insecure network
- Authentication takes the form of temporary tickets. Password is never send over network.
- Only ever have to enter password once (per session)
- Really freaking complicated
  - Socratic Dialogue of Kerberos
  - ELI5: Kerberos

BeyondCorp and Zero Trust

- The traditional firewall + privileged intranet model doesn't really work nowadays: remote workers, PaaS, networked apps
- Google's done some work talking about a new approach with dynamic policies and more fine-grained access controls.
- Identifying access by user and device info and other heuristics
- Yes, this shit is buzzwordy as hell

Encryption

- Everyone has secrets and something to hide: bank accounts, darknet purchases, desperate 2am Tinder messages, etc
  - If you claim other, would you unlock your phone and let me look through it?*
- Merely hiding things is not secure. Anything that is hidden can be found**.

* For a less glib treatment of this argument, see Daniel Solove's essay *'I've Got Nothing to Hide' and Other Misunderstandings of Privacy*

** This is security by obscurity

Encryption

Instead, let's "hide" things by converting information into something unreadable, except by you - that way, even if it's "found," it's useless, because it's meaningless garbage

"secret" -> "fc683cd9ed1990ca2ea10b84e5e6fba048c24929"

There's two parts, encrypting and decrypting

"Fc683cd9ed1990ca2ea10b84e5e6fba048c24929" -> ???

Symmetric Encryption

- A single key used to encrypt and decrypt data.
- Similar weaknesses as passwords
  - need to share secret with counterparty
  - distributing keys itself needs to be secure
- Fast (compared to asymmetric cryptography)
- Current standard is AES (*Advanced Encryption Standard*), certified by NIST and used by US FedGov, NSA, etc. to protect government secrets.

Asymmetric Encryption

Problem: How do we share keys over an insecure connection?

Asymmetric Encryption

Solution: Public Key Cryptography. There's two parts:

- Private: Keep secure, don't share it with anyone
- Public: Share as much as possible, announce to the world it's yours (Doing this is actually a really hard problem)

Anything encrypted with the public key can *only* be decrypted with the private key. Nothing secret needs to be securely sent beforehand like symmetric cryptography.

Asymmetric Encryption: RSA

RSA is the classic example

- Relies on the fact that factoring big numbers is hard
- It's getting easier to do so and keys keep getting bigger
- We're technically screwed if quantum computing becomes a thing, but it's supposed have been ten years away for 30 years now.

Asymmetric Encryption, ECC

Elliptic Curve Cryptography (ECC)

- Much smaller keys
- Way more complicated. What the hell is a “safe” curve?
- Relies on the fact that the elliptic curve discrete logarithm problem is intractable
- Concerns that the NSA put a backdoor in NIST’s recommendations for curves

Any Questions?

Hashing

- How do we make sure the large OS from a sketchy mirror doesn't have a backdoor in it?
- How do we securely store passwords?

Hashing! Turn any data (even that bloated as hell 20GB Windows 10 OS) into a small fixed string. This is

- Fast
- Deterministic

Hashing, cont.

There are lots of hashing / digest algorithms, you may have heard of some of them, e.g.

1. MD5 (broken, don't use)
2. SHA1 (broken by Google in 2017), SHA2, SHA256, SHA512
3. Keccak (aka SHA3)

man sha{1,256,4512,224,384}sum and md5sum to see how to use these on the terminal

Hashing, cont.

All of them turn arbitrary data into fixed-size outputs with critical cryptographic properties met:

1. **Preimage resistance** (given h, it's hard, i.e. heat death of the universe will happen first, to find m such that $h = \text{hash}(m)$)
2. **Second preimage resistance** (given m_1, finding m_2 such that $\text{hash}(m_1) = \text{hash}(m_2)$ is hard)
3. **Collision resistance** (finding any m_1, m_2 such that $\text{hash}(m_1) = \text{hash}(m_2)$ is hard)

Practical Hashing: Verifying Integrity

- A malicious user can’t change the original file and still have the same hash output.
- You download a large file from a mirror and guarantee integrity by checking the hash against a trusted source

Practical Hashing: Password Verification

Never store user passwords in plain (or encrypted) text.

- Hash them with a slow function (bcrypt, argon2, PBKDF) and hash the new password to see if they're the same.
  a. Now ~~if~~ when you lose passwords, malicious attackers can't get them, because hash functions aren't reversible.
  b. Why slow? Prevent brute-forcing attacks. A legitimate user should only have the hash run once (or at least ten times if you're me)

Practical Hashing: Password Verification

Recall that hashing functions are deterministic and *public*.

I promise you there's a lookup table matching the top 1000 passwords to their bcrypt values floating around somewhere.

How do we defeat this?

Practical Hashing: Password Verification

Salt your passwords:

- Append a random value to your hash input (e.g. username) to change the output (by a lot).
- (hunter2, bcrypt(hunter2)) is probably out in the wild, but (hunter2 + mdcha, bcrypt(hunter2 + mdcha)) isn't.

Any Questions?

Signatures and Certificates

The public key cryptography system can also be used the verify identities.

Suppose I want to use your public key to verify your identity. I can encrypt something with it, and ask you to decrypt it and show me the correct value. If you can decrypt the value, you must own the private half of the key and can be authenticated.

Signatures and Certificates

Suppose you want to prove that a message sent by you is actually sent by you. You can use your private key to “sign” the message by encrypting it, and your public key can be used to decrypt the signature to verify that you (identified by your published public key) did in fact send the message, since only you and not an adversary would have the corresponding private key.

Signatures and Certificates

How can we trust that the public key we get belongs to the user?

If I gave you a public key and told you that it belonged to your bank, how do you verify this?

This is a nontrivial problem. You have to consider things like revocation and mistakes in transmission.

Signatures and Certificates, cont.

Enter certificates:

- A certificate is a cryptographically-signed message indicating trust in a public key
- Who signs the certificate? (i.e. who do we trust to tell us someone else is trustworthy?)
    - Luckily it's not turtles all the way down

Signatures and Certificates, cont.

- Root certificates: OS' include a number of root certificates that are the basis of trust over the network. Certificates are signed in a chain leading up to a root; if the chain is valid, the cert at the end is presumed to be trusted.
- Not a foolproof system (*cough* wosign/diginotar *cough*)

  https://packages.debian.org/stretch/ca-certificates

Any Questions?

File Security

File Permissions and Ownership

Background

- UNIX is a multi-user environment
- Everyone has things to hide
- If multiple people can login but you have files you need to keep hidden (e.g., your private keys), you need some permissions and ownership setup to let you and only you access the files

- UNIX authentication paradigm involves users and groups:
  - Users can be characterized by individual logins, e.g. abizer or www-data
  - Users can have permissions controlled collectively by means of groups, e.g. the sudo group lets users in that group use the sudo command.

See also: Role-based Access Control

UNIX Permissions Model

Everything is a file, every “file” owned by a user and a group

3 components in UNIX permission model: for file’s

- owning user
- owning group
- everyone else (other)

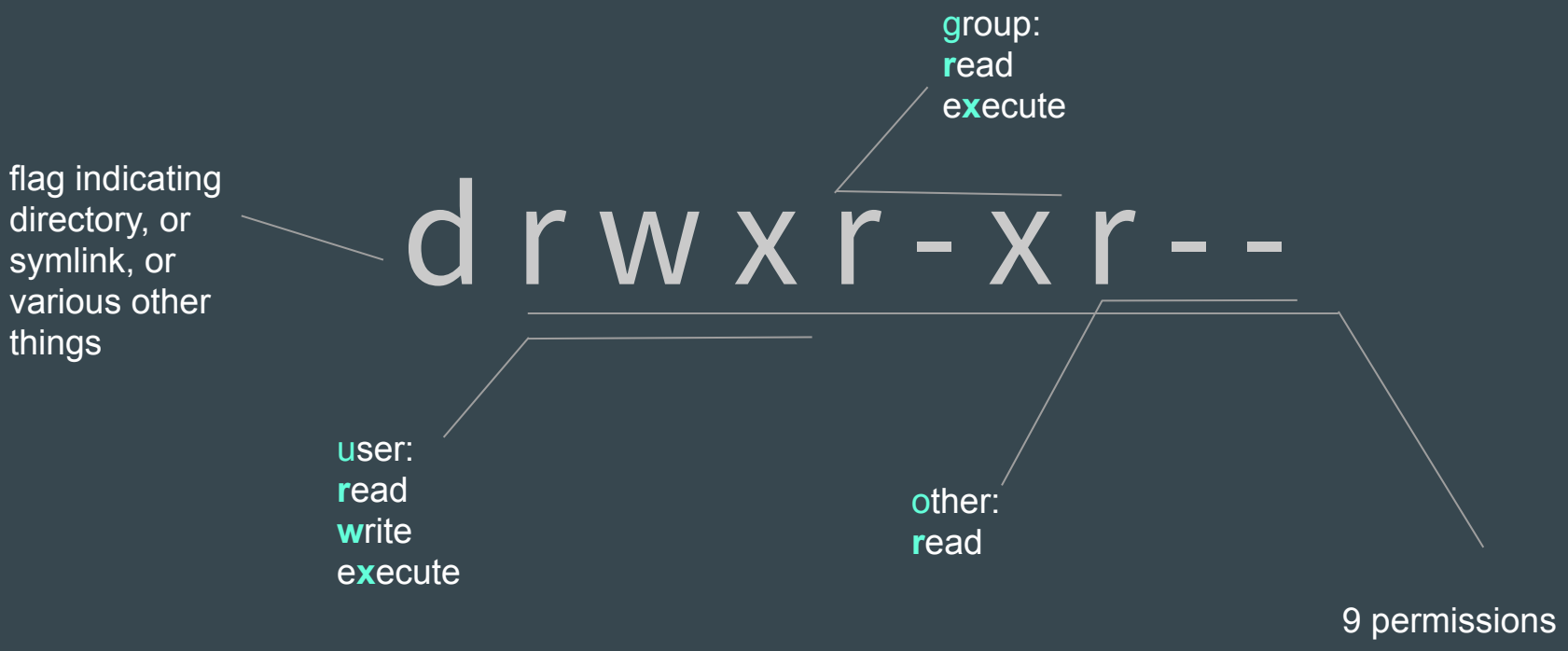
3 types of permissions:

- read
- write
- execute

9 possibilities: user rwx, group rwx, other rwx

```
admin@staff:~$ ls -la
total 112
drwxr-xr-x 11 admin admin  4096 Oct 28 19:12 .
drwxr-xr-x  5 root  root   4096 Oct  2 16:49 ..
drwxr-xr-x  2 admin admin  4096 Sep 21 21:11 .augeas
-rw-------  1 admin admin 32058 Oct 28 20:16 .bash_history
-rw-r--r--  1 admin admin   220 May 15 12:45 .bash_logout
-rw-r--r--  1 admin admin  3526 May 15 12:45 .bashrc
drwx------  3 admin admin  4096 Oct 17 02:08 .cache
drwx------  3 admin admin  4096 Sep 17 12:02 .config
drwxr-xr-x  4 admin admin  4096 Oct  3 12:47 .gem
drwxr-xr-x  2 admin admin  4096 Oct  3 12:46 .nano
-rw-r--r--  1 admin admin   675 May 15 12:45 .profile
drwxr-xr-x  4 admin admin  4096 Sep 17 14:23 .puppet
drwx------  2 admin admin  4096 Sep 17 12:09 .ssh
drw-r--r--  3 admin admin  4096 Oct  3 12:38 test
drwxr-xr-x  2 admin admin  4096 Oct 17 01:21 .vim
-rw-------  1 admin admin 19814 Oct 28 19:12 .viminfo
```

Explanation of Permissions



Permissions Masks

4: Read (100)

2: Write (010)

1: Execute (001)

= 7 (111)

What is:

1. 644
2. 755
3. 400
4. 777
5. rwxr-xr-x
6. rw-r--r--

Modifying Permissions

2 primary ways to modify permissions/file access:

- Change file ownership
- Change file permissions directly

Changing File Ownership

- Suppose a file you need to modify is owned by root
  - e.g. you tried to do `$ sudo cp foofile barfile`
- Changing file permissions won't do you any good, because the file's owning user and group will still be root and root.
- Need to change file's ownership: enter the chown command
- Syntax is simple: `[sudo] chown [-R] newuser:newgroup [FILES]`

```
abizer@lowgpa:~/a8/chown$ ls -lA
total 4
-rw-r----- 1 root root 18 Mar 16 00:23 important.pwd
abizer@lowgpa:~/a8/chown$ cat important.pwd
cat: important.pwd: Permission denied
abizer@lowgpa:~/a8/chown$ sudo chown abizer:ocf important.pwd
abizer@lowgpa:~/a8/chown$ cat important.pwd
s3cr37 p48600w0rd
abizer@lowgpa:~/a8/chown$ ls -lA
total 4
-rw-r----- 1 abizer ocf 18 Mar 16 00:23 important.pwd
```

Changing File Permissions

- Suppose you only need to read the file, doesn't matter if root owns it.
- Note that the "other" bits are all turned off - suppose we just added an "other" read permission instead?
- This is the chmod command:
  - `[sudo] chmod [-R] [permissions] [FILES]`

```
abizer@lowgpa:~/a8/chmod$ ls -lA
total 4
-rw-r----- 1 root root 16 Mar 16 00:41 important.pwd
abizer@lowgpa:~/a8/chmod$ cat important.pwd
cat: important.pwd: Permission denied
abizer@lowgpa:~/a8/chmod$ sudo chmod o+r important.pwd
abizer@lowgpa:~/a8/chmod$ ls -lA
total 4
-rw-r--r-- 1 root root 16 Mar 16 00:41 important.pwd
abizer@lowgpa:~/a8/chmod$ cat important.pwd
s3cr3t p4$$w0rd
```

Why is this important?

In a practical sense, bad file security is one of the easiest ways to leak information or give an attacker far too much privilege on your system.

Suppose you are using public-key crypto to encrypt your files on a multi-user system. What happens if you set this as the permissions on your private key?

```
-rwxrwxrwx 1 admin admin 20 Oct 31 16:49 rsaprivate.key
```

Anyone on your system can read your key and decrypt your files!

What happens if a program running as root gets exploited? Whole system is compromised!

File Security

- Remember the *principle of least privilege*: Don't chmod 777 your private key
- Make sure that your programs don't execute with more permissions than necessary, e.g. don't run your IRC client as the root user
- Make sure your files have their permissions appropriately limited, e.g., don't leave your passwords world-readable

File Security

- Don’t randomly sudo shit because your command failed.
- Don’t just do your daily tasks as root.
- Don’t upload your AWS creds onto GitHub.
- Don’t upload your API tokens onto GitHub.
- Don’t upload your SSH private key on GitHub.

set[ug]id

setuid and setgid are auxiliary bits that can be set in the permissions of a file that allow it to execute as the owning user/group respectively, regardless of which user calls it

e.g. allow anyone to execute a single script, which would run as (for example) the root user, or as the www-data group, to achieve some necessary end

Mandatory / Role-Based Access Control

- User/group paradigm doesn't offer that great flexibility
- Enter things like AWS IAM: Actions require permissions, which can be grouped into roles, and users can be assigned roles
- On an action, we check if the user is in that role
- Attribute-based access control can be extremely powerful, but extremely complicated as well

Any Questions?

Network Security

Protecting a computer and applications over the network

Security implications of networked software

Background

- On an isolated system, your threat surface is limited to anyone who can get physical access to the system
- On a networked system, anyone on the network can access (and attack) your system
  - Security becomes much harder if you have to be connected to the public internet

Therefore, it is important to minimize your attack surface and reduce attack vectors as much as possible

Basics

- Recall SSH - secure shell, allows you to log in to a remote system over the network
- Suppose a vuln. exists in sshd that allows randos to log in as the root user
  - byebye secure files

Basics

- Suppose you fix the SSH vulnerability, but have an FTP server running as root on the system, and this gets compromised
  - byebye secure files
- Suppose you leave a test user with a weak password in a VM given to you by a DeCal...
  - byebye secure files
  - hello random bitcoin miner

Basics

- Suppose you fix everything, but a user starts a web server and serves a program that has a directory traversal vulnerability
  - Attacker simply traverses back to read /etc/shadow and gets your (hashed) password
  - byebye secure files (after some period of time for offline password cracking)

Network Security is Hard.

Types of network attacks

There are lots of ways to attack networked systems:

Wiretapping, Man in the Middle, Denial of service, Application vulnerabilities such as buffer/heap overflow, SQL injection, directory traversal, CSRF, SSRF, XSS, worms, rootkits, spam, cryptomining, ransomware, phishing, dozens more...

So what do we do about... wiretapping?

- Use encryption to protect data in transport, via e.g. TLS (transport layer security).
- In that vein, have your websites be served over HTTPS.
  - **Especially** if you're handling sensitive credentials.
  - It's not that hard. Let's Encrypt let's you do it for free.
- Enforce HSTS (Requires clients to interact with your web server via HTTPS, preventing downgrade attacks)

So what do we do about... man-in-the-middle?

- Always verify identities: SSH fingerprints, TLS certificates
- Be vigilant when connecting to new systems. Don't ignore certificate validation errors!
- Certificate Pinning: Only ever trust certs with/signed by those hashes, not any cert signed by any root CA
  - Still needs TOFU, revocation is still a mess.
- Certificate Transparency: public auditing system for issued certificates (voluntary, requires auditors, not always used)

So what do we do about... denial of service?

- Distribute critical services over larger footprint: servers, geography, etc
- Use filters to reduce ability of attacker to overwhelm you.
- Outsource network protection to a CDN.
- Enforce quotas for users
- Don't have any vulnerable apps amplifying the attack
- There's a lot of different defenses.

So what do we do about... application vulnerabilities?

- Always keep applications up to date with security patches
- Avoid running multiple services on the same critical system to prevent one application vulnerability from causing data loss from other services.
  - Our mail server (anthrax) is one VM and our IRC server is another (flood) and our log server (democracy) is another, etc.
- Don't run applications as root if at all possible

So what do we do about... application vulnerabilities?

- Keep things updated—The possibility that you're targeted by a zero-day is fairly low, the possibility that you get targeted by some random dork with a botnet looking for published vulnerabilities is much higher.
  - Or just don't have a million and a half add-ons/plugins that you don't need—minimize attack surface
- It's never a bad idea to be hella aggressive with your firewall. Services like shodan.io/masscan crawl the entire internet looking for easy pickings.
  - In fact, some dipshit neo-nazi can even just send garbage to internet-connected printers if you're not careful!

Knowledge is Power!

The most important defense is being aware of potential threats in the first place:

- Read sources like OWASP Top Ten and be aware of current threats
- Use security mailing lists of software you use. (e.g. debian-security-announce)
- Security conferences provide contemporary information for security
  - Read some USENIX Security Symposium / IEEE Security and Privacy / ACM CCS papers for fun!
  - Watch Black Hat / DEFCON talks
- Do red team exercises like OverTheWire, HackThisSite, and OWASP's Juice Shop. They're fun and you can get a more concrete understanding of what attacks can comprise of.

Logging

- The only way to respond to attacks is with good logs. Or else you're just guessing.
- You can see if someone is aggressively scanning your servers.
- Keep a centralized logging server using something rsyslog or syslogd: it's less likely an attacker can escape a distributed logging system after breaking into a singular machine.
- Breaking into a system can be easy, breaking into a system unnoticed is much, much harder.

Backups!

- Remember threat modeling: It's much less expensive to hit by ransomware if you can just restore data from an hour ago.
- See B9: Version Control and Backups for more information and best practices

Resources

- Security Engineering by Ross Anderson
  - I love this book and it's free. Fuck you, Pearson and Cengage
- Security - Arch Wiki
  - "If the Arch Wiki had dating advice, no Linux user would be single." - Ex-SM of OCF
- OWASP Top Ten Project
- EFF's Security Education Companion and Security Self-Defense
- USENIX Best Papers
- Interactive: OverTheWire, HackThisSite, and OWASP's Juice Shop.
- Transcript of Snowden's Q&A at IETF 93

Contact: mdcha@berkeley.edu, IRC: dongkyun

Discussion: TLS MiTM

Is this attack possible?

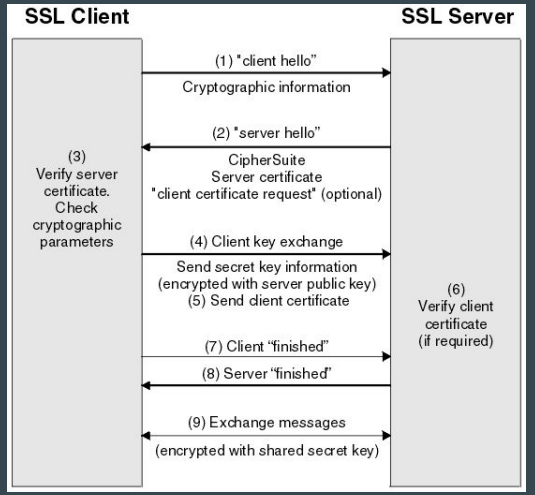
1. Mallory poisons DNS to redirect eve.com to Mallory's server
2. Bob attempts to visit eve.com, and starts a TLS handshake
3. Mallory proxies data between Bob and eve.com, runs Diffie-Hellman in both directions, decrypting and re-encrypting data as it flows between them
4. Bob thinks he's connecting to eve.com, eve.com thinks she's only connecting to Mallory
5. Mallory can read all the secret data between Bob and eve.com

Does this attack work, why or why not? Does TLS have any defense against this type of attack? How would it protect against this type of attack?

How does it work together?

Example: secure web browsing via TLS

1. Establish TCP connection
2. Establish TLS connection
    a. Client requests server's certificate and verifies the signature on it - warns you if certificate validation fails
    b. Client and server hash some secret values and perform Diffie-Hellman key exchange to establish a shared secret to bootstrap symmetric key encryption to protect the TLS session
3. Enter password into login page
    a. Password is protected over the wire by TLS, then, the server hashes it using a (hopefully) secure hashing algorithm and compares the hash against the hash in the database, then allows you to log in



Don't ignore certificate validation errors!